Instructions for using Microsoft Visual Studio 2015 with the CCI Configuration POC

June 9, 2016

Table of Contents:

Introduction

This document describes how to use MSVS2015 with CCI. Refer to cci/msvc10/cci/README.txt in the github report for general Windows instructions.

Step-by-step instructions are provided for the following:

1. Building the three Proof of Concept (PoC) libraries
   - cciapi.lib
   - ccibrokerimpl.lib
   - cciparamimpl.lib
2. Running the batch mode regression test suite (verify.pl)
   - Consisting of both user and developer examples
3. Building and running the examples inside Microsoft Visual Studio 2015

Prerequisites

Tested tool versions are noted below only for reference; other versions may work fine.

1. SystemC Installation

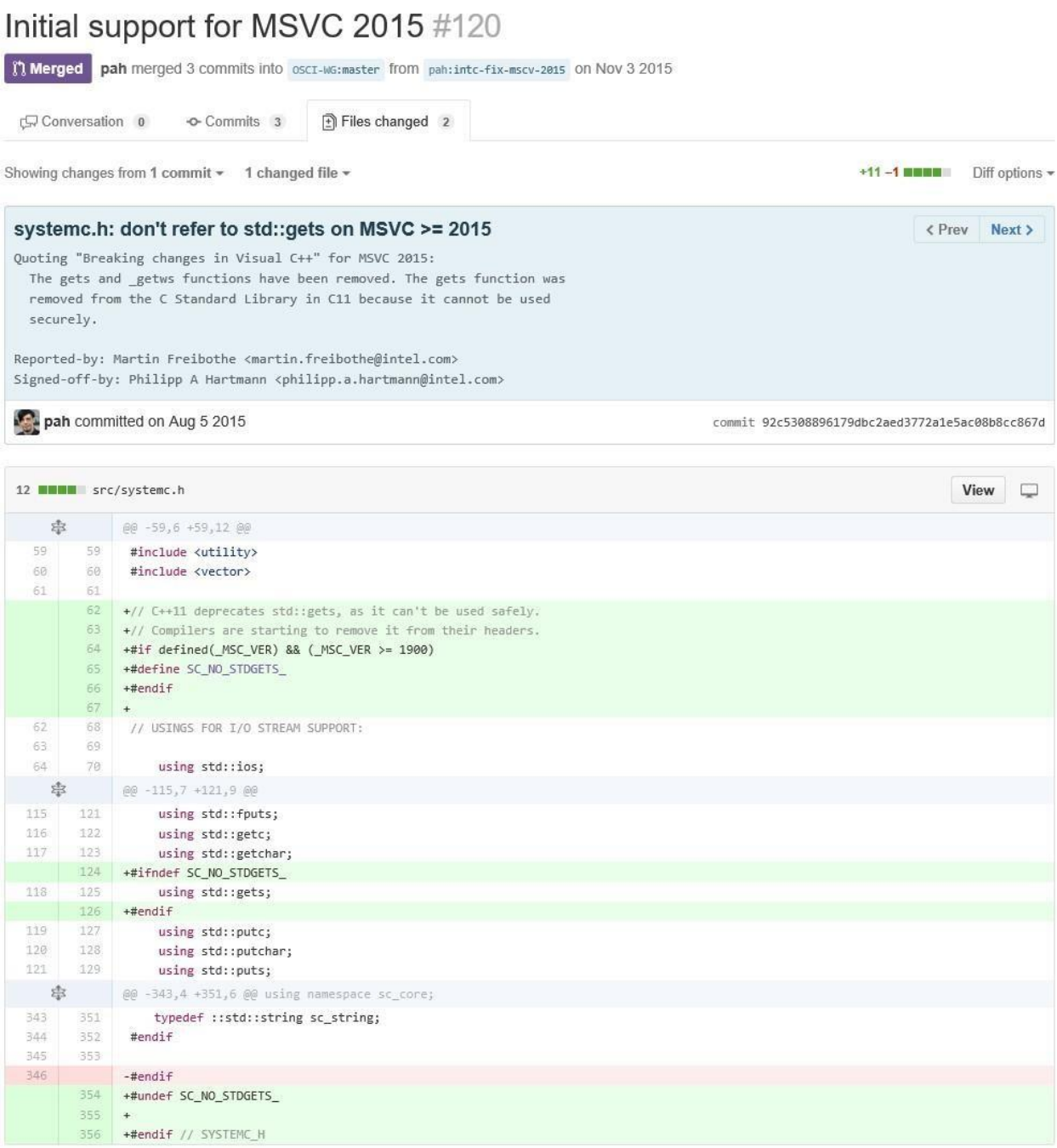
- Download the 2.3.1 version of 'Core SystemC Language and Examples' from: http://accellera.org/downloads/standards/systemc.
- Copy the msvc10 directory to msvc14 and build the SystemC library for desired configuration (e.g. x64 Debug).

2. Initialize a github workspace
   - Install github windows client https://help.github.com/articles/set-up-git/
   - Refer to Getting started in the CCI WG.docx for creating a fork and cloning therepository

3. Install MSVS2015SP1
   - https://www.visualstudio.com/en-us/downloads/download-visual-studio-vs.aspx
4. Install Cygwin Perl Package and Cygwin diffutils package for 64 bit Installation from https://cygwin.com/; the following versions weretested:
   - Cygwin 2.5.1-1
   - Perl 5.22.2-1
   - Diffutils 3.3-3
5. Install Boost Library, version; 1.57.0 wastested
   - Follow the steps from http://slu.livejournal.com/17395.html
   - Update boost\config\compiler\visualc.hpp to support latest version ofMSVS2015SP1 The highlighted change below is required to avoid the 'Unknown Compiler version' warnings. The current version of MSVS2015 (SP1) we are using is Microsoft (R) C/C++ Optimizing Compiler Version 19.00.23918 for x64.

```
// last known and checked version is 19.00.23918 (VC14 CTP4):
#if (_MSC_VER > 1800 && _MSC_FULL_VER > 19002391 8)
#  if defined(BOOST_ASSERT_CONFIG)
#     error "Unknown compiler version - please run the configure tests and report the results"
#  else
#     pragma message("Unknown compiler version - please run the configure tests and report the results")
#  endif
#endif
```

6. Since MSVS 2015 does not support 'gets' , update systemc.h



Initial support for MSVC 2015 #120

Merged pah merged 3 commits into OSCI-WG:master from pah:intc-fix-mscv-2015 on Nov 3 2015

Conversation 0 Commits 3 Files changed 2

Showing changes from 1 commit 1 changed file +11 −1 Diff options

systemc.h: don't refer to std::gets on MSVC >= 2015 Prev Next

```
Quoting "Breaking changes in Visual C++" for MSVC 2015:
  The gets and _getws functions have been removed. The gets function was
  removed from the C Standard Library in C11 because it cannot be used
  securely.

Reported-by: Martin Freibothe <martin.freibothe@intel.com>
Signed-off-by: Philipp A Hartmann <philipp.a.hartmann@intel.com>
```

pah committed on Aug 5 2015 commit 92c5308896179dbc2aed3772a1e5ac08b8cc867d

12 src/systemc.h View

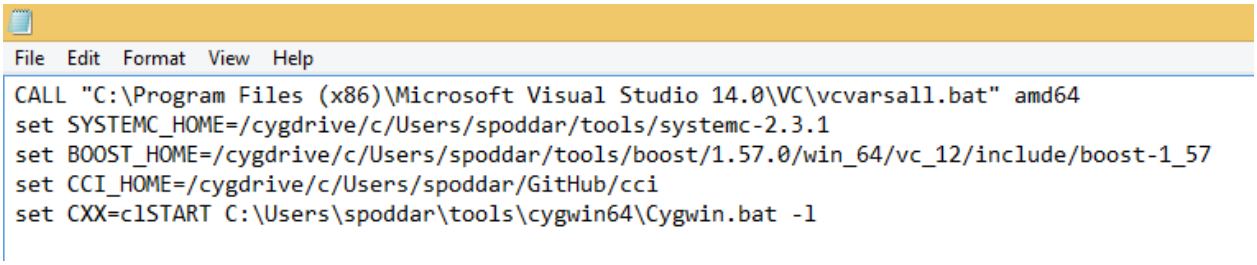
```
@@ -59,6 +59,12 @@
 #include <utility>
 #include <vector>

+// C++11 deprecates std::gets, as it can't be used safely.
+// Compilers are starting to remove it from their headers.
+#if defined(_MSC_VER) && (_MSC_VER >= 1900)
+#define SC_NO_STDGETS_
+#endif
+
 // USINGS FOR I/O STREAM SUPPORT:

     using std::ios;
@@ -115,7 +121,9 @@
     using std::fputs;
     using std::getc;
     using std::getchar;
+#ifndef SC_NO_STDGETS_
     using std::gets;
+#endif
     using std::putc;
     using std::putchar;
     using std::puts;
@@ -343,4 +351,6 @@ using namespace sc_core;
 #endif

-#endif
+#undef SC_NO_STDGETS_
+
+#endif // SYSTEMC_H
```

Steps

Step1: Build the three PoC libraries

1. Copy the ‘msvc10’ directory to the SystemC architecture alias ‘msvc14’ for Microsoft Visual Studio 2015
2. Initialize environment variables

   The project setup uses environment variables to reference SystemC, Boost and CCI directories. These can be set in System Settings or using a Visual Studio launch batch file as shown below:

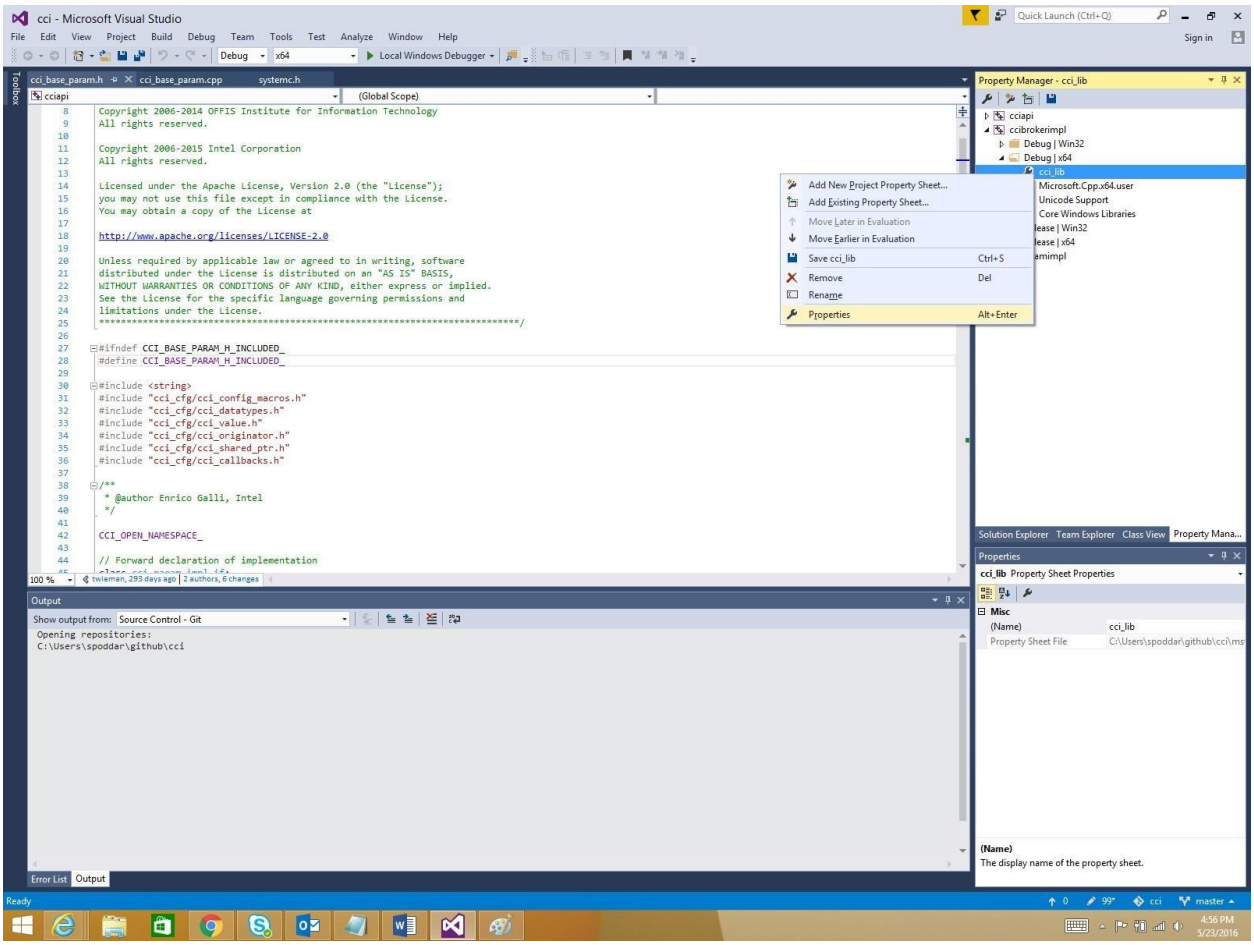


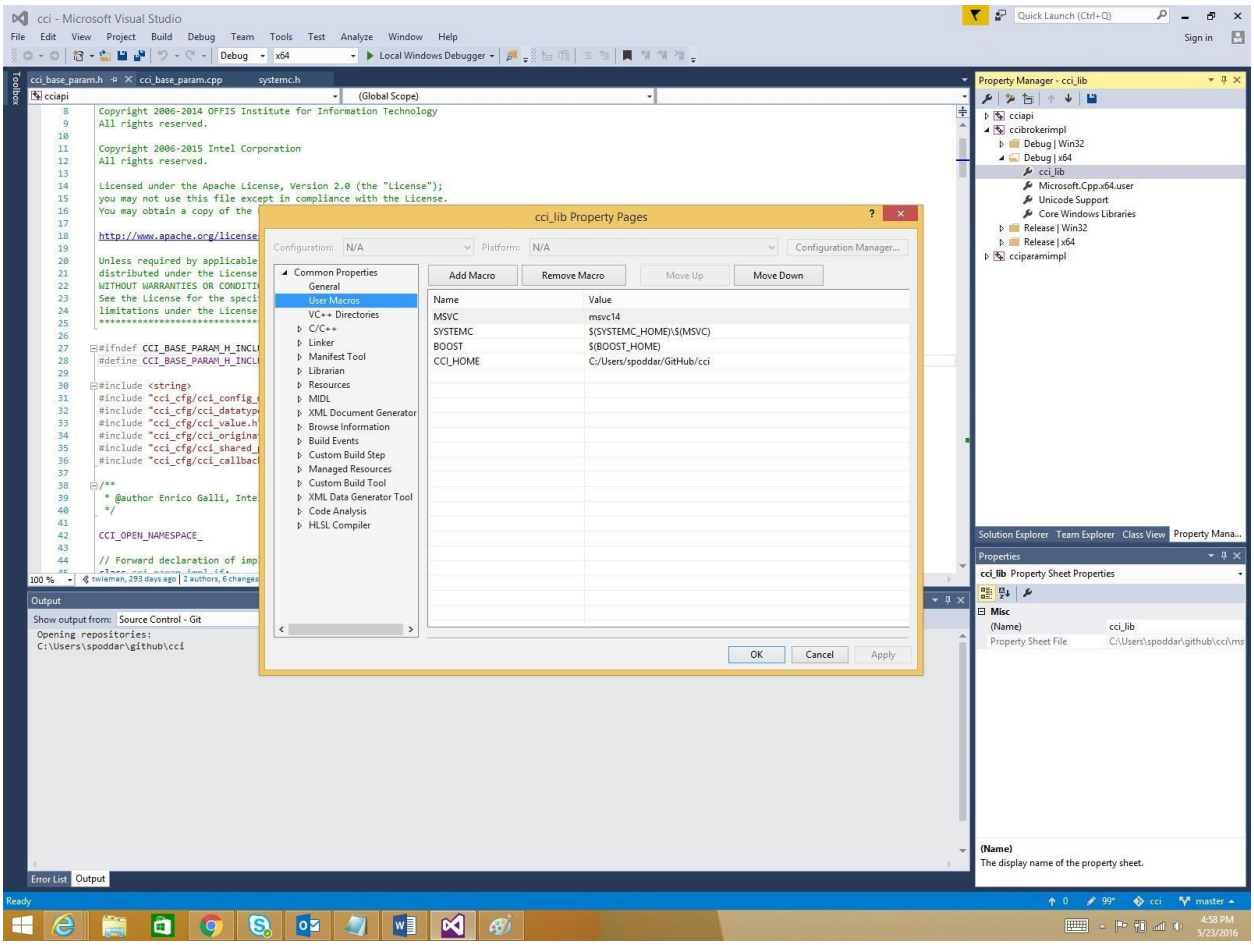
3. Update the CCI libraries property sheet (msvc14/cci/cci_lib.props)
   - Change the MSVC macro from “msvc10” to “msvc14”.

   All three projects share this property sheet which specifies compiler and linker paths for SystemC and Boost using environment the variables $SYSTEMC_HOME and $BOOST_HOME.

   Open the existing MSVC Solution
   - File-> Open-> Project/Solution
   - From your CCI home -> msvc14 ->cci ->cci.sln
   - Under the ‘cci.sln’ solution, you can see three MSVC Projects:
     - cciapi
     - ccibrokerimpl
     - cciparamimpl

   You can view the common settings from the Property Manager by right clicking on the cci_lib property sheet for any of the CCI libraries and selecting “Properties”.


cci - Microsoft Visual Studio
File Edit View Project Build Debug Team Tools Test Analyze Window Help
Debug x64 Local Windows Debugger
cci_base_param.h cci_base_param.cpp systemc.h
cciapi (Global Scope)
Copyright 2006-2014 OFFIS Institute for Information Technology
All rights reserved.
Copyright 2006-2015 Intel Corporation
All rights reserved.
Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at
http://www.apache.org/licenses/LICENSE-2.0
Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
#ifndef CCI_BASE_PARAM_H_INCLUDED_
#define CCI_BASE_PARAM_H_INCLUDED_
#include <string>
#include "cci_cfg/cci_config_macros.h"
#include "cci_cfg/cci_datatypes.h"
#include "cci_cfg/cci_value.h"
#include "cci_cfg/cci_originator.h"
#include "cci_cfg/cci_shared_ptr.h"
#include "cci_cfg/cci_callbacks.h"
/**
* @author Enrico Galli, Intel
*/
CCI_OPEN_NAMESPACE_
// Forward declaration of implementation
Add New Project Property Sheet...
Add Existing Property Sheet...
Move Later in Evaluation
Move Earlier in Evaluation
Save cci_lib Ctrl+S
Remove Del
Rename
Properties Alt+Enter
Property Manager - cci_lib
cciapi
ccibrokerimpl
Debug | Win32
Debug | x64
cci_lib
Microsoft.Cpp.x64.user
Unicode Support
Core Windows Libraries
Solution Explorer Team Explorer Class View Property Mana...
Properties
cci_lib Property Sheet Properties
Misc
(Name) cci_lib
Property Sheet File
(Name)
The display name of the property sheet.
Output
Show output from: Source Control - Git
Opening repositories:
C:\Users\spoddar\github\cci
Error List Output
Ready
master
4:56 PM
5/23/2016

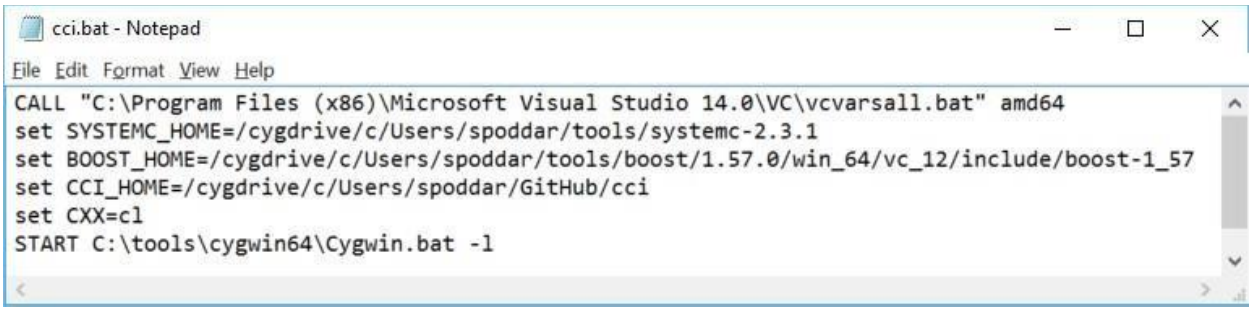


4. Build the Solution
   - Choose the configuration
     - Debug / Release
     - Win32 / x64
   - Build the whole cci solution
   - All three CCI PoC libraries will be created for the given configuration.

Step2: Run regression tests

The UNIX regression script requires a UNIX environment, such as Cygwin.

1. Initialize environment
   The key environment variables need to be initialized in a manner consistent with our Visual Studio setup. This can be done in a Cygwin shell launch script as shown below, or simply included in your shell startup settings (i.e. .bashrc).

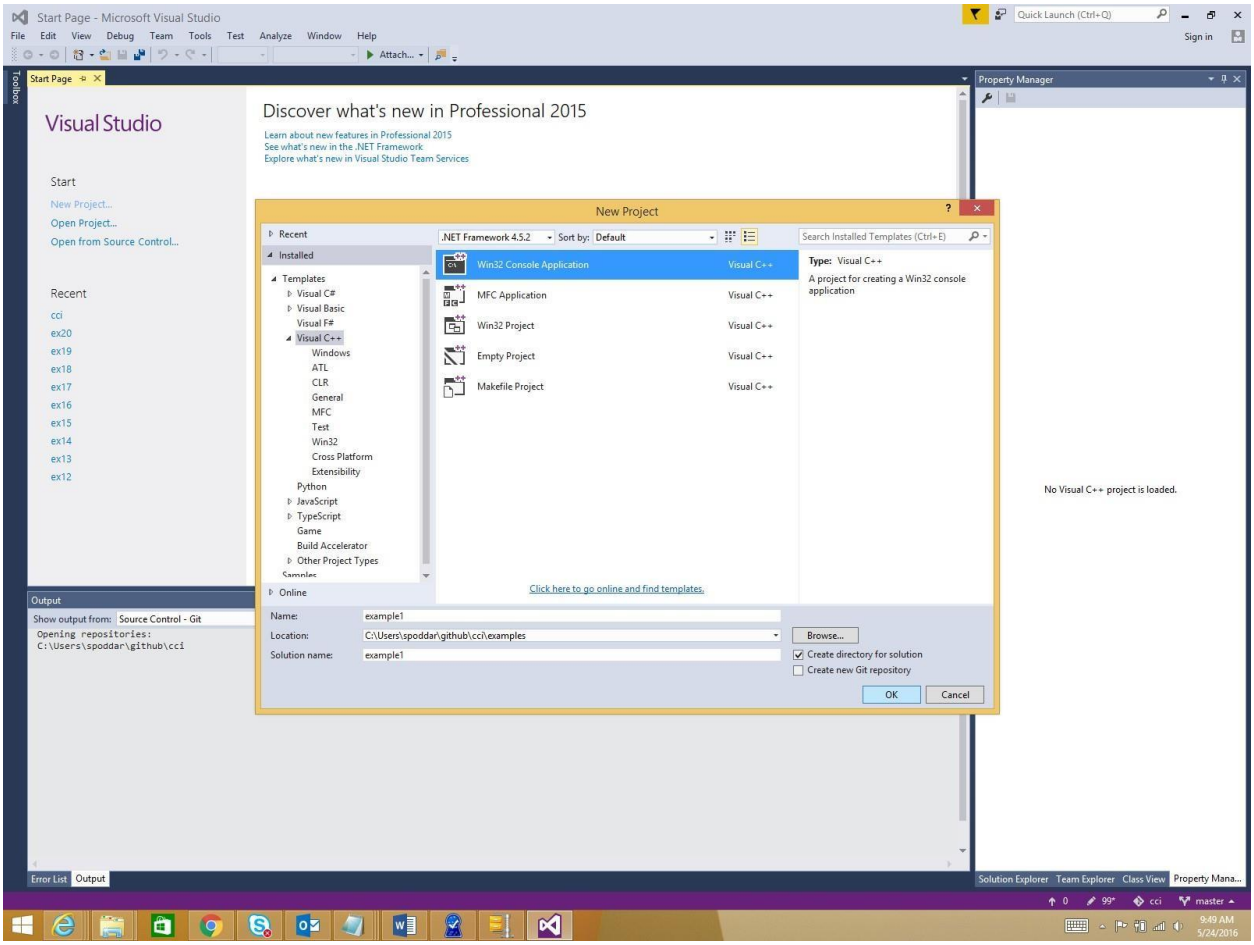


2. Start a properly initialized Cygwin Shell
   Note: if link.exe is found from the Cygwin installation (which results in linker failures) instead of the Visual Studio installation, a path adjustment will be necessary:
   $ PATH=/cygdrive/c/Program\ Files\ \(x86\)/Mirosoft\ Visual\ Studio\ 14.0/VC/bin/amd64:$PATH
3. Create a directory (under cci/) to hold the regression results and cd into it
   Ex. mkdir run
   cd run
4. Run the verify.pl script as described in cci/examples/README.txt
   Ex. ../scripts/verify.pl -g dev_examples examples (runs all CCI examples using debug library)
   Note: add the verbose flag (-v) to see compiler/linker commands.
5. Compare against the expected results in cci/scripts/results/summary.txt

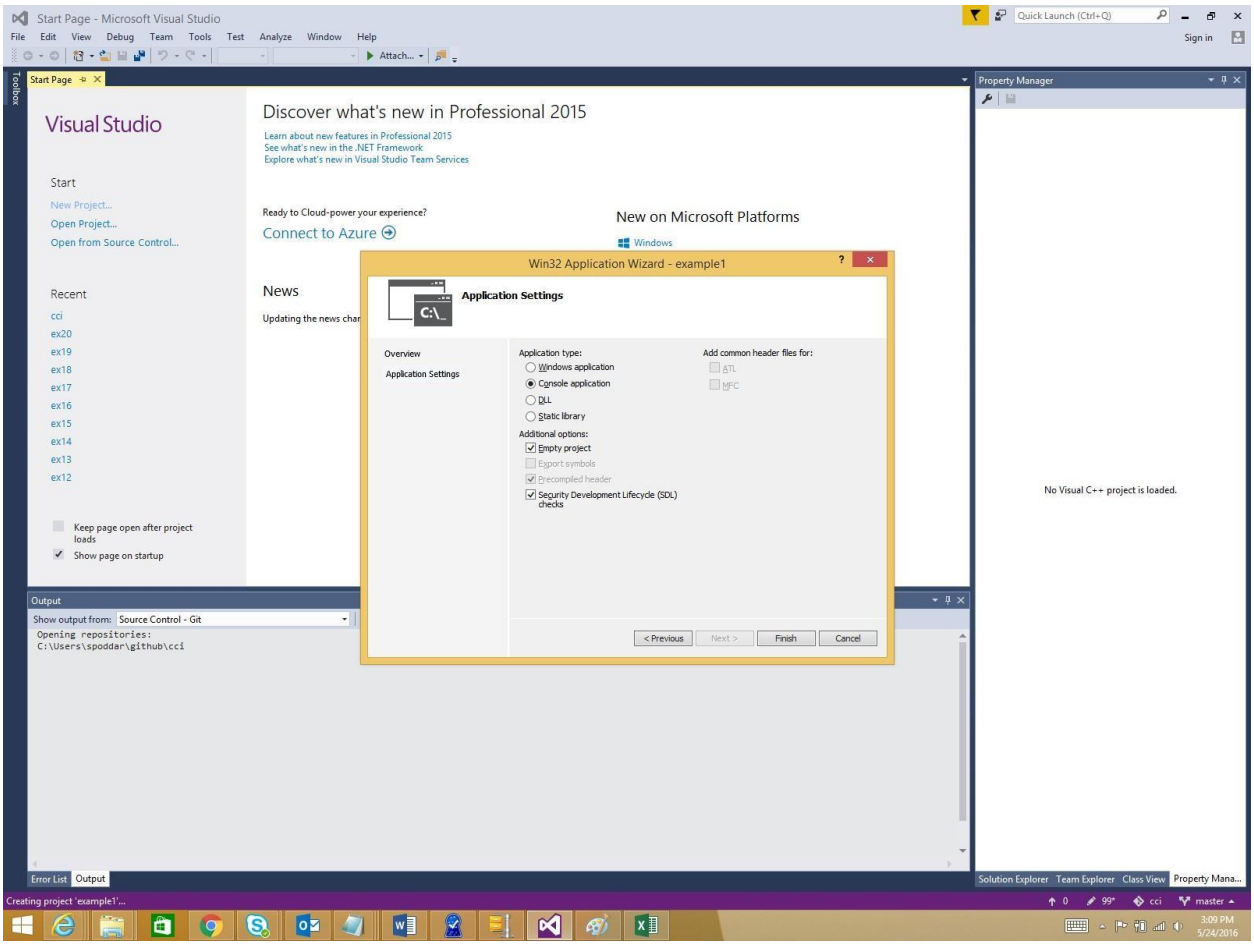
---

Step3: Building MSVC Projects for CCI applications

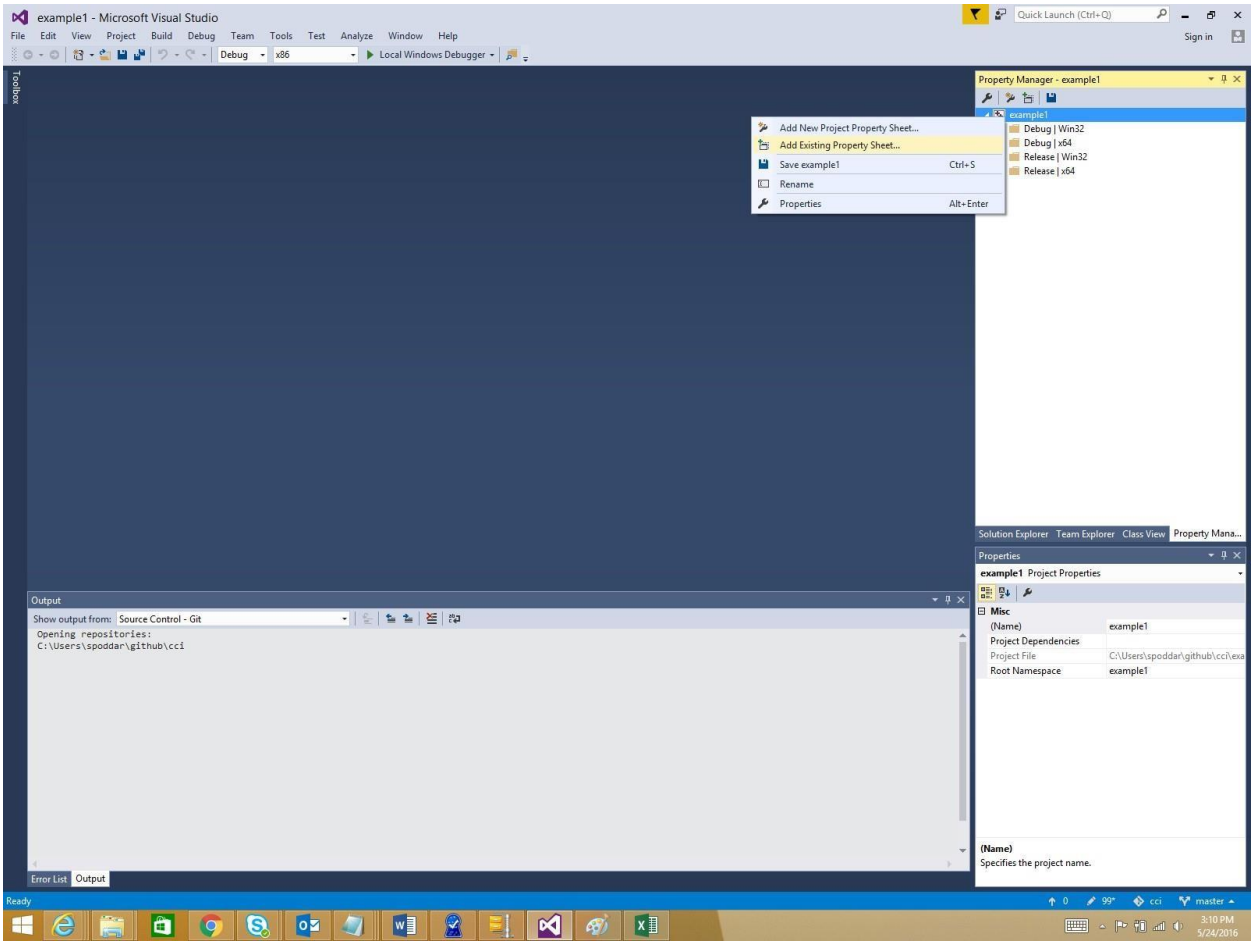
1. Open your Microsoft Visual Studio 2015 ccisolution
2. File-> New-> Project
   Select Visual C++ in the left pane
   Select Win32 Console Application in the middle pane
   Name: ex01_proj
   Location:C:\Users\spoddar\github\cci\msvc14\cci\ (same as for CCI libs)
   Solution: Add to solution (to have all CCI projects together)
   Hit the OK button.



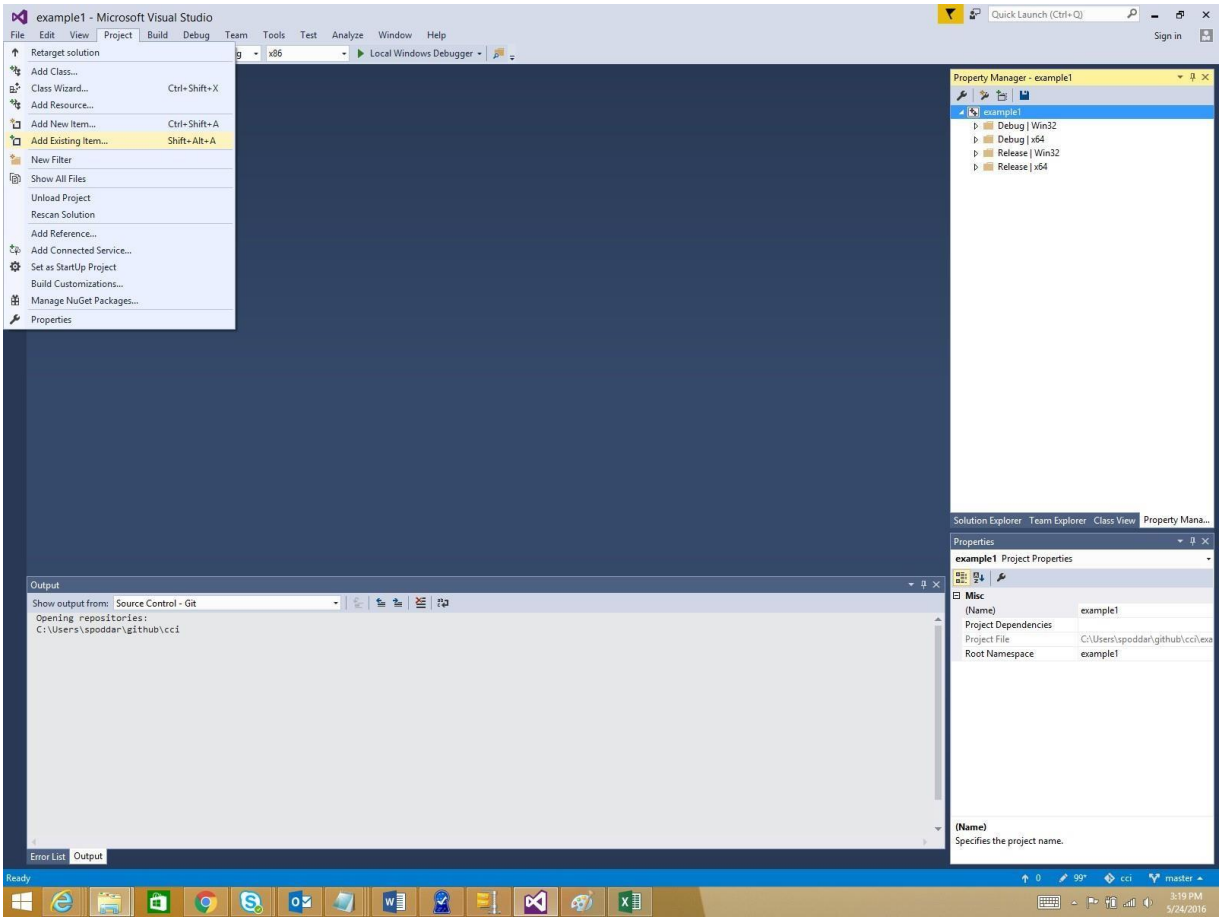
3. Hit Next on the Win32 Application Wizard
   Select Application Type: Console Application
   Additional options: Empty project
   Hit Finish



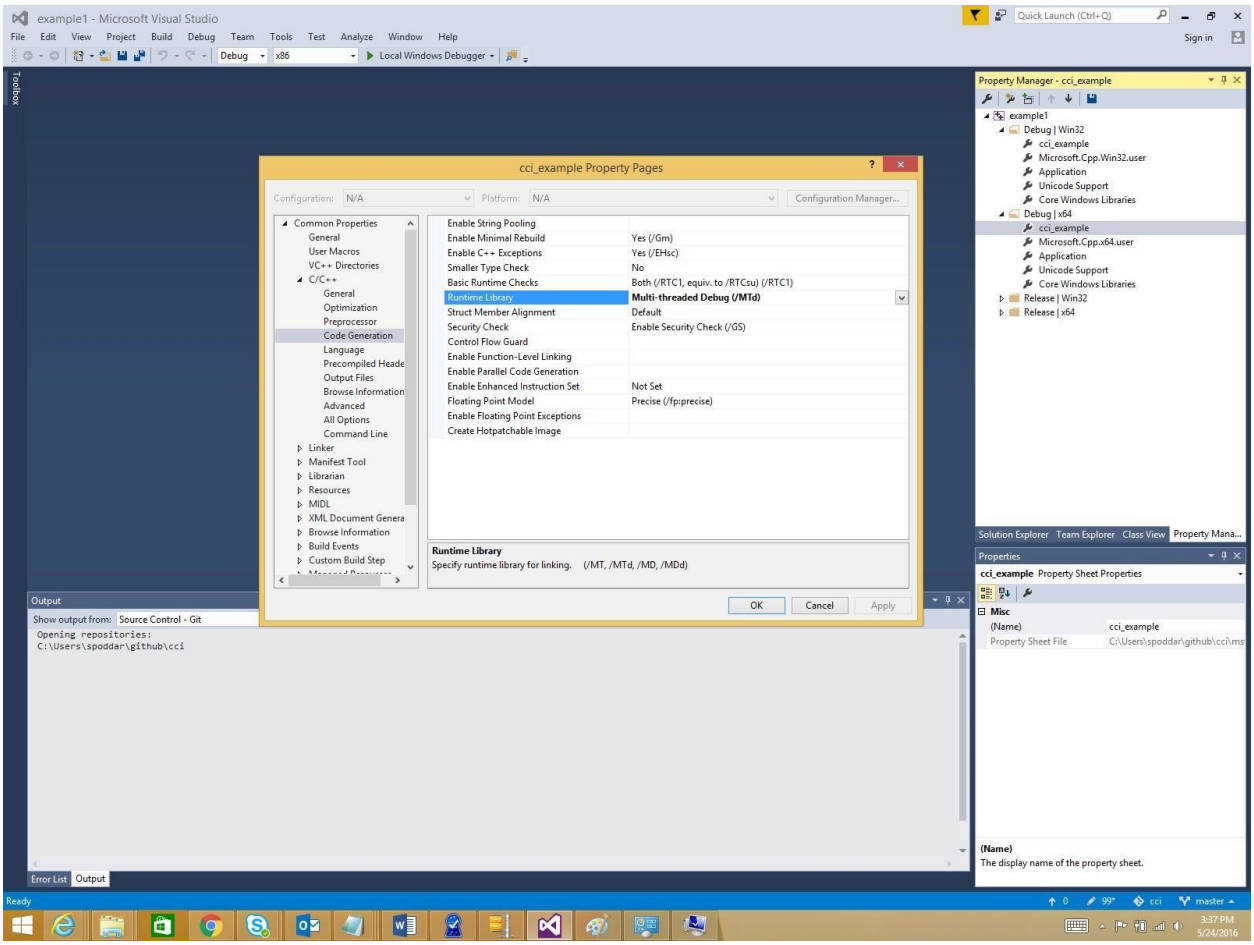
4. Go to Property Manager-> right click example1-> Add existing property sheet
   Select cci_examples.props from msvc14\cci
   Example C:\Users\spoddar\github\cci\msvc14\cci\cci_examples.props.



5. Return to Solution Explorer and select Project->Add existing Item
    a. Navigate to the corresponding example directory (e.g. C:\Users\spoddar\github\cci\examples\ex01_Simple_Int_param) and select all *.h and *.cpp files then hit the Add button.
    Note: For gs_example_diff_impl among dev_examples, remember to add all *.h and *.cpp files from the subdirectory param_impl.



6. Verify your configuration (e.g. x64-Debug). From the Solution Explorer right click on the project and Select Properties. Navigate to Configuration Properties->C/C++->Code Generation-> Runtime Library and choose the same runtime library used to build your SystemC (e.g. /MTd).



You are now ready to build and run/debug.